MQTT Tutorial

Message Queue Telemetry Transport

# Theoretical Background

This tutorial should outline the different requirements for IoT (Internet of Things) devices compared to commonly used distributed computing environments and how these requirements will be addressed by MQTT.

MQTT is a lightweight protocol which focuses on the special requirements of IoT. The following list gives an idea about the essential requirements of IoT that are fulfilled by MQTT:

* Bidirectional **event-driven** communication (asynchronous over callbacks)
* **No open ports** on IoT devices (reduce attack surface)
* **Session awareness** (if connection was interrupted, there is no need to send meta-data again)
* **Simplicity** (IoT devices do not have a lot of memory and computing power)
* Use the message queuing **publish & subscribe** approach
* **Reduce** protocol **overhead** to a minimum
* **Detect** **ungracefully** device **disconnects** (Some IoT devices are powered by battery and the server [named broker] should be informed immediately if a node is offline)

This tutorial was inspired by HiveMQ <http://www.hivemq.com/mqtt/>

## How does MQTT work?

MQTT consists of three different types of actors:

**MQTT Broker** is a server which is hosted in the cloud

**MQTT Publisher** is a IoT device that produces data and submits it to the broker

**MQTT Subscriber** is a device or application which is interested in the data that an IoT device has published.

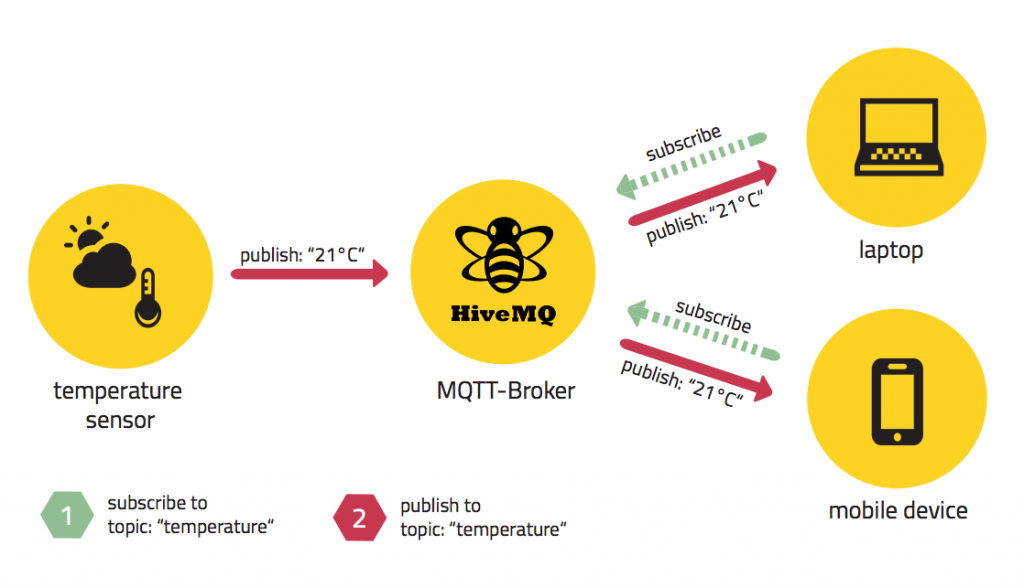


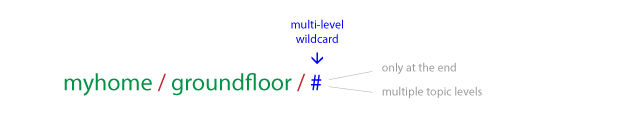
Figure 1: Example of a simple MQTT Architecture (HiveMQ, 2017)

## How is data being structured within the protocol

We do not want to receive every message that is sent via MQTT. Thus, we need a structure within the protocol that organizes and divides our messages. The message organizing approach used in MQTT is topic-based.

MQTT messages will be published in a chosen topic. The topic itself is nothing more than a simple string, which represents a structure. Subscriber will only be able to receive the data if they subscribed to the same topic. If a subscriber wants to receive all data of a topic and its subtopics wildcards can be used.

|  |  |
| --- | --- |
| Wildcard | Purpose |
| # | Get messages from all subtopics (multi-level) |
| + | Mask only one level of the subtopics (one-level) |



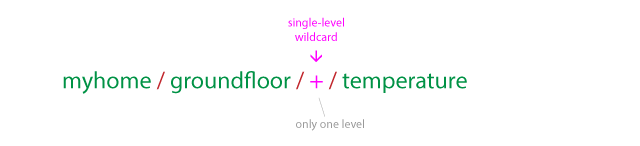
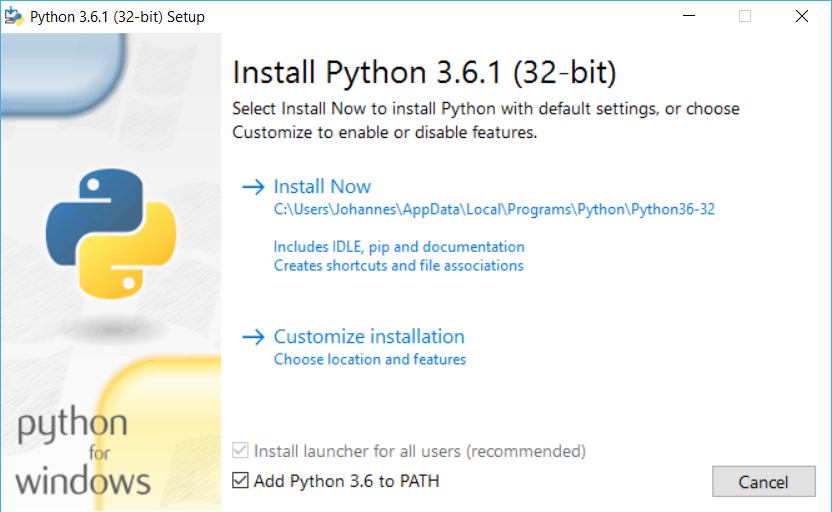


Figure 2: MQTT Topic Wildcars (HiveMQ, 2017)

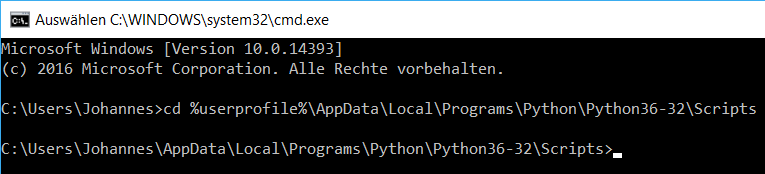
# Requirements for the Tutorial

* **Install Python 3.6.x** (Important download the 3.6 Version of Python and not the 2.7!)  
  <https://www.python.org/downloads/>
  + Start the Installation
  + Important: Select Checkbox **“Add Python 3.6 to PATH”**



* **Install PAHO MQTT Library for Python**
  + Open Windows Command Line and type in the change directory command

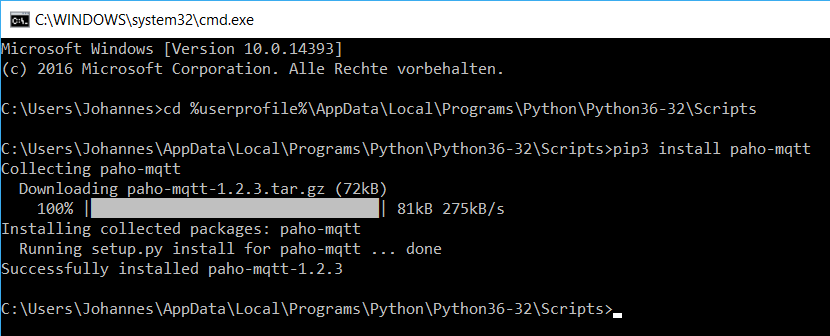
*cd %userprofile%\AppData\Local\Programs\Python\Python36-32\Scripts*



* + Type in *pip3 install paho-mqtt*



* + If everything was successfully installed the should be written on your screen



# Connect to a Broker

HiveMQ is a german platform which hosts an MQTT broker. We will use this MQTT broker to send data from a Sensor Node (Publisher) to a Subscriber.

Host: **broker.hivemq.com**  
Port: **1883**

MQTT is supported by a variety of different programming languages and operating systems. These are only a few examples of Libraries available for MQTT:

* [Android: Paho Android Service](http://www.hivemq.com/blog/mqtt-client-library-enyclopedia-paho-android-service)
* [Java: Eclipse Paho](http://www.hivemq.com/blog/mqtt-client-library-encyclopedia-eclipse-paho-java)
* [Objective C: MQTT-Client-Framework](http://www.hivemq.com/blog/mqtt-client-library-encyclopedia-mqtt-client-framework)
* [C: Paho C](http://www.hivemq.com/blog/mqtt-client-library-pahocclient)
* [C (Embedded): Paho Embedded](http://www.hivemq.com/blog/mqtt-client-library-encyclopedia-paho-embedded)
* [Arduino: Arduino Pubsub Client](http://www.hivemq.com/blog/mqtt-client-library-encyclopedia-arduino-pubsubclient/)
* [C#: M2Mqtt](http://www.hivemq.com/blog/mqtt-client-library-encyclopedia-m2mqtt)
* [Javascript: MQTT.js](http://www.hivemq.com/blog/mqtt-client-library-mqtt-js)
* [Python: Paho Python](http://www.hivemq.com/blog/mqtt-client-library-paho-python)
* [Go: Paho Go](http://www.hivemq.com/blog/mqtt-client-library-encyclopedia-golang)

You can read details about MQTT and the connection details at this page: <http://www.hivemq.com/try-out/>

# Publish Sensor Data

The sensor node publishes the temperature in a given interval (10 seconds) to the topic “uno/8380/insertyourname/temperature”.

The following code should be saved in a file called publisher.py and do not forget to substitute insertyourname with your name!

**import** paho**.**mqtt**.**client **as** paho

**import** time

**def** getTemperature**():**

#in a real-world application this function would read the temperature from a Sensor

temperature **=** 72

**return** temperature

**def** on\_connect**(**client**,** userdata**,** flags**,** rc**):**

**print(**"CONNACK received with code %d." **%** **(**rc**))**

**def** on\_publish**(**client**,** userdata**,** mid**):**

#mid is the messageId

**print(**"mid: "**+**str**(**mid**))**

#Instancing a MQTT client

client **=** paho**.**Client**()**

#Specify Callback Functions

client**.**on\_publish **=** on\_publish

client**.**on\_connect **=** on\_connect

#Initiating the connection

client**.**connect**(**"broker.mqttdashboard.com"**,** 1883**)**

client**.**loop\_start**()**

**while** **True:**

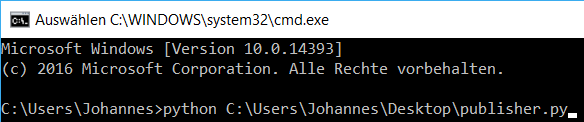
#This is an endless loop that will read data from the sensor every 10 seconds and sends it to the broker

temperature **=** getTemperature**()**

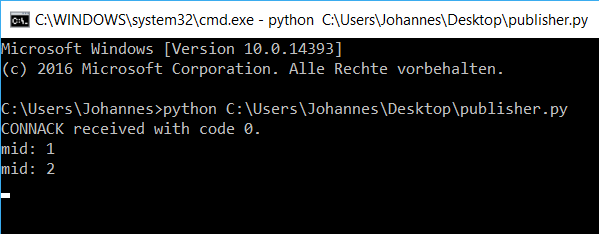
**(**rc**,** mid**)** **=** client**.**publish**(**"uno/isqa8380/insertYourName/temperature"**,** str**(**temperature**),** qos**=**1**)**

time**.**sleep**(**10**)**

To execute the script open the windows command line type in python and drag and drop the script into the command line. Press enter to execute the script.



The publisher now starts to send data to the message broker. Please leave the script running in the background and continue with the subscribe script.



# Subscribe to Sensor Data

The on\_message function will be triggered as soon as the subscriber receives data from the broker. The on\_message function takes care about the logic how the data will be processed. In our case we simply want to print the message to see it on our screen.

The following code should be saved in a file called subscriber.py and do not forget to substitute insertyourname with your name!

**import** paho**.**mqtt**.**client **as** paho

**def** on\_subscribe**(**client**,** userdata**,** mid**,** granted\_qos**):**

**print(**"Subscribed: "**+**str**(**mid**)+**" "**+**str**(**granted\_qos**))**

**def** on\_message**(**client**,** userdata**,** msg**):**

**print(**"Message Received:"**)**

**print(**msg**.**topic**+**" "**+**str**(**msg**.**qos**)+**" "**+**str**(**msg**.**payload**))**

#the subscriber could have here more logic to use the data ..

#Instancing MQTT client

client **=** paho**.**Client**()**

#Defining Callback Functions

client**.**on\_subscribe **=** on\_subscribe

client**.**on\_message **=** on\_message

#Connect to Broker

client**.**connect**(**"broker.mqttdashboard.com"**,** 1883**)**

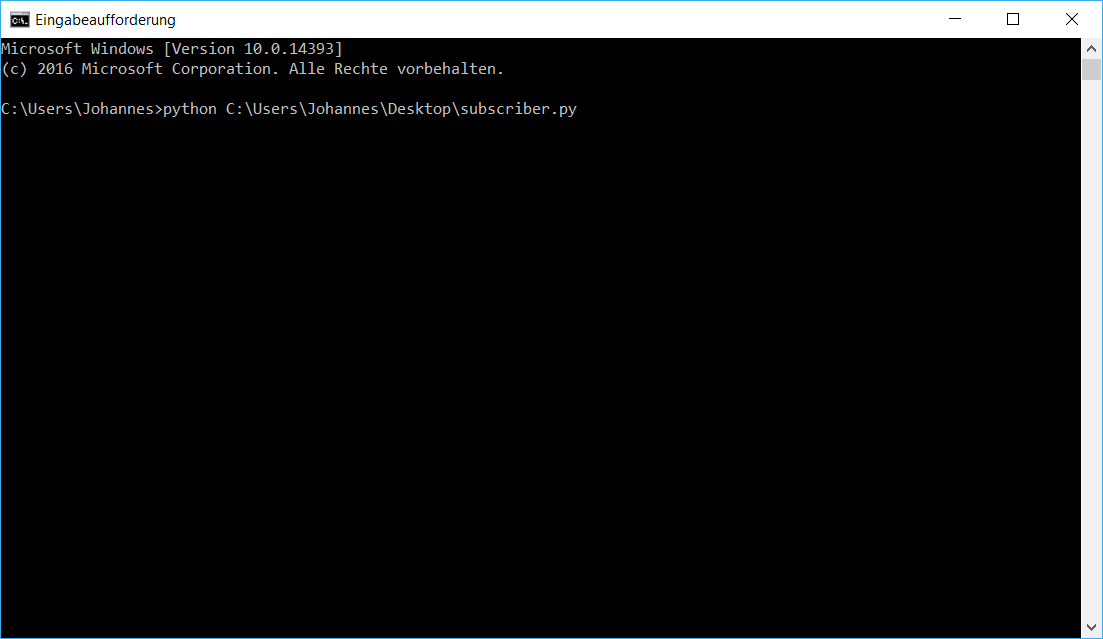
#We are subscribing to the topic 8380 and all subtopics (the # sign tells the broker we are interested in subtopics too)

client**.**subscribe**(**"uno/isqa8380/insertYourName/#"**,** qos**=**1**)**

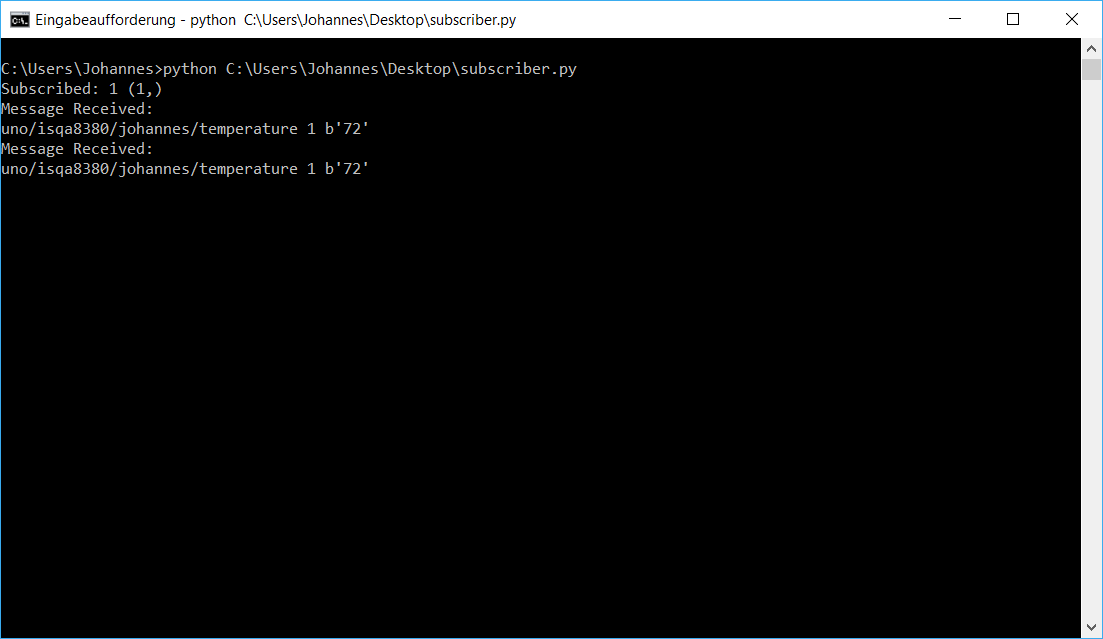
#Lets program running

client**.**loop\_forever**()**

To execute the script open another windows command line and drag and drop the script into the command line.



Press enter to execute the script. The subscriber script should now show you arriving messages.



**References**:

HiveMQ (2017) *Fundamentals about MQTT* available at <http://www.hivemq.com/blog> accessed 05/17/2017

This Tutorial was written by Johannes Hofer for ISQA8380

[johanneshofer@outlook.com](mailto:johanneshofer@outlook.com)